Knowledge-Based Approach for Generating Target System Specifications from a Domain Model

Hassan Gomaa, Larry Kerschberg, and Vijayan Sugumaran

Center for Software Systems Engineering
Department of Information and Software Systems Engineering
George Mason University
Fairfax, Virginia 22030-4444

Abstract

Several institutions in industry and academia are pursuing research efforts in domain modeling to address unresolved issues in software reuse. To demonstrate the concepts of domain modeling and software reuse, a prototype software engineering environment is being developed at George Mason University to support the creation of domain models and the generation of target system specifications. This prototype environment, which is application domain independent, consists of an integrated set of commercial off-the-shelf software tools and custom-developed software tools. This paper describes the knowledge-based tool that has been developed as part of the environment to generate target system specifications from a domain model.

Keywords: domain modeling, reuse, software engineering environments, object repository, requirements elicitation, knowledge-based tool support.

INTRODUCTION

An application domain is defined to be a collection of systems that share common characteristics. A domain model is used to capture common characteristics and variations among a family of software systems in a given application domain. From the domain model, a target system can be generated by tailoring the domain model according to the requirements of the target system. Thus, a target system engineer can develop the specification for a target system in terms of the domain model specified previously by a domain analyst, and does not have to perform a full systems analysis every time a new target system has to be constructed.

At George Mason University, a project is underway to support software engineering lifecycles, methods, and prototyping environments to support software reuse at the requirements and design phases of the software lifecycle, in addition to the coding phase. A reuse-oriented software lifecycle, the Evolutionary Domain Lifecycle (Gomaa, 1989; Gomaa, 1991a) has been proposed, which is a highly iterative lifecycle that takes an application domain perspective allowing the development of families of systems. A domain analysis and modeling method has also been developed (Gomaa, 1990). This paper describes a knowledge-based approach for generating target system specifications from a domain model.

DOMAIN ANALYSIS AND MODELING

The Evolutionary Domain Life Cycle (EDLC) Model (Gomaa, 1989) is a software lifecycle model that eliminates the traditional distinction between software development and maintenance. Instead, systems evolve through several iterations. Hence, systems developed using this approach need to be capable of adapting to changes in requirements during each iteration. Furthermore, because new software systems are often outgrowths of existing ones, the EDLC model takes an application domain perspective allowing the development of families of systems.

Parnas referred to a collection of systems that share common characteristics as a family of systems (Parnas, 1979). According to Parnas, it is worth considering a family of systems

when there is more to be gained by analyzing the systems collectively rather than separately, i.e., the systems have more features in common than features that distinguish them. The concept of viewing an application domain as consisting of a family of systems has been adopted by various researchers (Batory, 1989; Kang, 1990; Pyster, 1990; Lubars, 1989).

When considering the development of a family of systems, it is necessary to replace the traditional system development activities of Requirements Analysis, Requirements Specification, and System Design with activities that span the entire application domain. These are Domain Analysis, Domain Specification, and Domain Design.

A Domain Model is a problem-oriented architecture for the application domain that reflects the similarities and variations of the members of the domain (Gomaa 1992). Given a domain model of an application domain, an individual target system (one of the members of the family) is created by tailoring the domain model according to the requirements of the individual system.

A Domain Model is initially created by means of a Domain Analysis. Domain Analysis (Prieto-Diaz, 1987) is a requirements analysis of a family of systems for a domain, rather than of a given target system. A domain analysis must address the requirements of the family of current systems as well as anticipate future changes. Although some future changes may be anticipated, it is unlikely that all future changes can be anticipated. It is therefore necessary for the Domain Model to be evolutionary. It needs to be capable of evolving as new (i.e., unanticipated) requirements are added and as existing requirements are changed in unanticipated ways. Domain Analysis is comparable to a systems analysis performed on a broader scale. It involves the analysis of existing target systems in the application domain as well as interviewing domain experts and capturing their knowledge of existing features, including known or anticipated variations in the domain.

Reuse is an important goal in domain modeling. A key aspect of this work is the way it combines generation technology with composition technology. Reuse by generation (Biggerstaff, 1987) implies a top-down approach in which a target system is generated from a domain model by tailoring the domain model according to the target system requirements. Reuse by composition (Biggerstaff, 1987) is a bottom-up approach in which components residing in a reuse library are located and reused, ideally without change. Instead of requiring software developers to search large reuse libraries, the domain model has an index into the reuse library, so that reusable software components may be more easily located and included in the target system implementation.

Multiple Views of Domain Model

Applying the domain modeling method, the application domain is modeled by means of the following views:

- *Aggregation Hierarchy.* The Aggregation Hierarchy is used to decompose complex, aggregate object types into less complex object types, eventually leading to simple object types at the leaves of the hierarchy.
- *Object Communication Diagrams.* Objects in the real world are modeled as concurrent processes that communicate with each other using messages. The object communication diagrams, which are hierarchically structured, show how objects communicate with each other.
- *State Transition Diagrams.* Because each active object is modeled as a sequential process, it may be defined by means of a finite state machine and documented using a state transition diagram.
- *Generalization / Specialization Hierarchies.* As the requirements of a given object type are changed to meet the needs of a given target system, the object type may be specialized by adding, modifying, or suppressing operations. The variants of a domain object type are stored in this hierarchy.
- *Feature / Object Dependencies.* This view shows for each feature (domain requirement) the object types required to support the feature.

The domain modeling method has been applied to developing a domain model for NASA's Payload Operations Control Center (POCC) Domain.

PROTOTYPE SOFTWARE ENGINEERING ENVIRONMENT

A prototype software engineering environment is being developed, which consists of an integrated set of software tools that support domain modeling and the generation of target system specifications. A schematic representation of the prototype environment is given in Figure 1. In order to expedite development of the prototype, the environment uses commercial off-the-shelf software as well as custom software. We are using Interactive Development Environments' Software Through Pictures CASE tool to represent the multiple views of the domain model, although semantically interpreting the views according to the domain modeling method. The information in the multiple views is extracted, checked for consistency, and mapped to an underlying representation, referred to as the domain specification, which is stored in an object repository (Gomaa, 1991b).

The domain specification stored in the object repository is augmented with domain features (requirements), inter-feature dependencies and feature/object dependencies. Inter-feature dependencies capture the relationships among features. For example, a feature may require the presence of some other feature(s). Another example of inter-feature dependency is that some features may be mutually exclusive or mutually inclusive. The feature/object dependencies relate features to objects, i.e., they define the object types required to support a particular feature. The domain analyst provides this feature-related information using the Feature Object Editor. Feature/object dependencies are stored in the object repository.

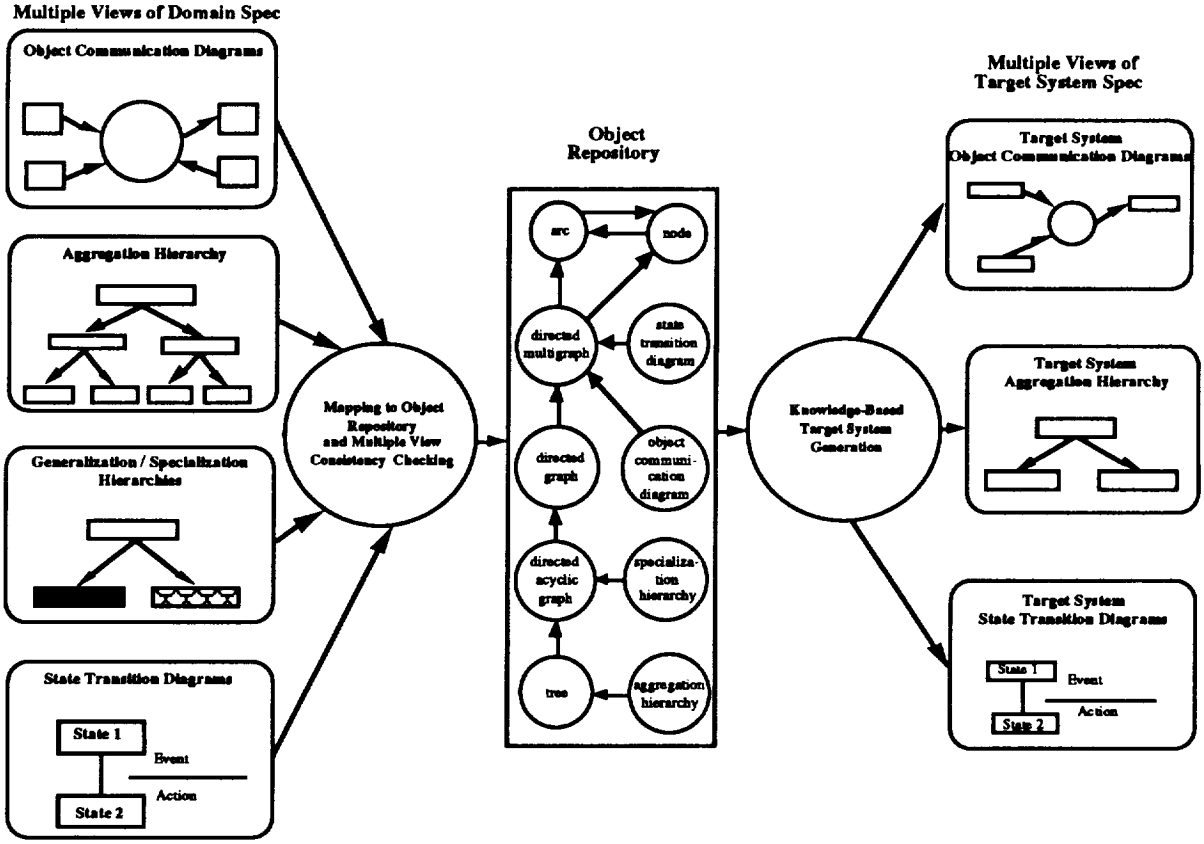

Figure 1. Prototype Domain Modeling Environment

The object repository interfaces with knowledge-based tools and provides the informal and formal specifications for reuse. Thus, the object repository provides a consistent domain model specification which can be accessed by various tools.

Once domain modeling is completed, the domain specification serves as the framework for generating target systems. The process of generating target systems from a domain model can be significantly improved with knowledge-based tool support. This tool not only must have knowledge about the domain model, but also must contain procedural knowledge about constructing target systems. A knowledge-based system called the Knowledge-Based Requirements Elicitation Tool (KBRET) is being developed to automate the process of generating the specifications for target systems. KBRET is used to assist with target system requirements elicitation and generation of the target system specification. This tool is implemented in the expert system shell CLIPS (C Language Integrated Production System), developed at NASA/Johnson Space Center (Giarratano, 1991). It conducts a dialog with the human target system requirements engineer, prompting the engineer for target system specific information. The course of the dialog is determined by the responses provided by the target system engineer. The output of this tool is used to adapt the domain model to generate the target system specification. When the target system objects have been assembled, the corresponding multiple views are derived by tailoring the multiple views of the domain model. The multiple views of the target system are then displayed using Software through Pictures.

The prototype software engineering environment is a domain-independent environment. Thus it may be used to support the development of a domain model for any application domain that has been analyzed, and to generate target system specifications from it.

KNOWLEDGE-BASED REQUIREMENTS ELICITATION TOOL (KBRET)

A target system specification is derived from the domain model by tailoring it according to the requirements specified for the target system. The process of generating a target system specification consists of gathering the requirements in terms of domain features, retrieving from the domain model the corresponding components to support those features, and reasoning about inter-feature and feature/object dependencies to ensure consistency. The Knowledge-Based Requirements Elicitation Tool (KBRET) facilitates the process of generating target system specifications from a domain model with multiple viewpoints.

The architecture of KBRET consists of two types of knowledge: domain-independent and domain-dependent knowledge. The domain-independent knowledge provides control knowledge for the various functions supported by KBRET. These functions include a browser, a feature selector, a dependency checker, and a target system generator. The domain-dependent knowledge represents the multiple views of an application domain model, including the feature/object dependencies. This knowledge is derived from the object repository through the KBRET Object Repository interface and structured as CLIPS facts (Sugumaran, 1991). The various components of KBRET are diagrammatically depicted in Figure 2.

The separation of domain-independent and domain-dependent knowledge is essential for providing scale-up and maintainability of domain specifications for large domains. Also, since the domain-independent knowledge is independent of the application domain, it can be used with domain-dependent knowledge from any application domain to generate target system specifications in that domain.

Domain Independent Knowledge

The domain-independent knowledge sources provide procedural and control knowledge for the various functions supported by KBRET. The *Dialog Manager* is responsible for carrying out a meaningful dialog with the target system engineer and eliciting the requirements for the target system. It addresses such issues as how, and in what sequence, the target system engineer should be prompted for various features, invoking and controlling the different phases of KBRET, the user interface, etc.

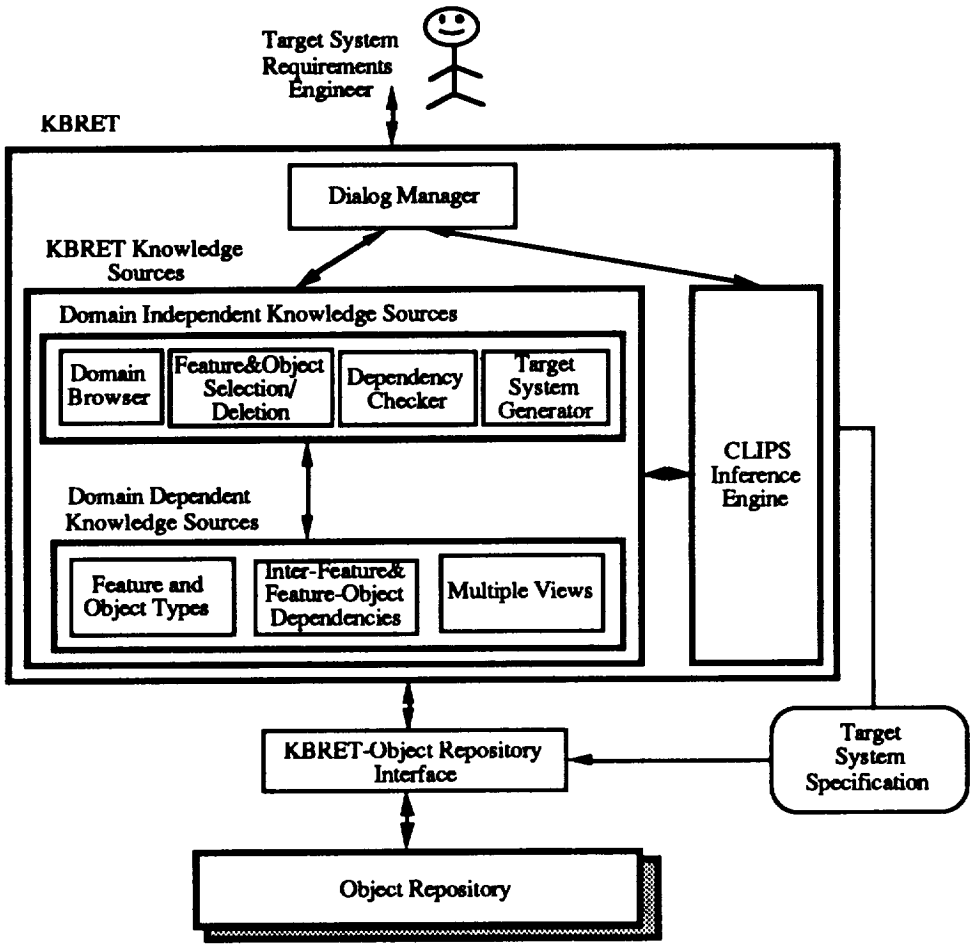

Figure 2. Knowledge Based Requirements Elicitation Tool (KBRET)

Before specifying the requirements for the target system, the target system engineer may wish to browse through portions of the domain model in order to gain understanding of the application domain under consideration. The *Domain Browser* knowledge source provides this facility. It provides rules for initiating and terminating the browsing facility and for accessing the appropriate domain-dependent knowledge sources.

The *Feature & Object Selection/Deletion* knowledge source keeps track of the selection or deletion of features for the target system and the corresponding object types. This knowledge source incorporates rules for selecting and deleting features and for invoking the appropriate rules for checking inter-feature and feature/object dependencies.

The *Dependency Checker* knowledge source cooperates with the Feature & Object Selection/Deletion knowledge source. When a particular feature is selected for the target system, the *Dependency Checker* enforces the inter-feature and feature/object dependencies for that feature. These dependencies are obtained from the *Inter-Feature & Feature- Object Dependencies* knowledge source, which is domain dependent, as shown in Figure 2. When a feature with some prerequisite features is selected, the *Dependency Checker* ensures that those prerequisite features are included in the target system. For example, in the POCC domain, the Verifying Real Time Commands feature requires the Sending Real Time Commands feature. If the Sending Real Time Commands feature is not selected and the

Verifying Real Time Commands feature is desired in the target system, the Sending Real Time Commands feature will be included in the target system before selecting the Verifying Real Time Commands feature.

Similarly, before deleting a feature from the target system, dependency checking is performed to ensure that it is not required by any other target system feature. Using the example from the previous paragraph, if both Sending Real Time Commands and Verifying Real Time Commands features are selected for the target system, the Sending Real Time Commands feature cannot be deleted from the target system as long as the Verifying Real Time Commands feature is selected for the target system. Thus, the *Dependency Checker* knowledge source has rules to enforce the inter-feature and feature/object dependencies so that a consistent target system is specified.

Once the feature selection for the target system is complete, the *Target System Generator* knowledge source begins the process of assembling the target system. The domain kernel object types are automatically included in the target system. Depending upon the features selected for the target system, the corresponding variant and optional object types are included according to the feature/object dependencies. The *Target System Generator* would detect if more than one variant (specialization) of a particular kernel or optional object type were included in the target system. These multiple variant object types have to be "integrated" to produce one integrated variant object type that would support the desired features in the target system. Some domains may require the presence of multiple variants of certain objects, and those variant objects should not be integrated. For example, in the POCC domain, multiple variants of observatory-related objects should not be integrated.

If multiple specializations of a particular kernel or optional object have been selected, and if they have to be integrated, the *Target System Generator* will access the *Multiple Views* domain-dependent knowledge source and check the appropriate generalization/specialization hierarchy to see if an integrated object type for those variant object types exists as a result of previous variant integration processes. If such an integrated object type is present, then that object type is included in the target system in lieu of those variant object types to be integrated. Once all the required integrated variant object types have been included, the target system generation is complete.

If an integrated variant object type is not in the domain model, the target system generation process is suspended until the domain analyst can specify the integrated variant and include it in the domain model. At that time the target system generation process can be reactivated to complete the specification of the target system.

Variant integration is a non-trivial task and may require considerable domain knowledge. Hence, completely automating the variant integration process will be a tremendous challenge.

Domain Dependent Knowledge

The domain-dependent knowledge sources contain specific information about a particular application domain. They are used by the domain-independent knowledge sources of KBRET in eliciting the requirements and generating the target system specification. The domain-dependent knowledge sources are derived from the domain specification, which is persistently stored in the object repository. The KBRET Object Repository Interface accesses the object repository and creates these knowledge sources using a representation that is compatible with the other knowledge sources of KBRET.

The *Features and Object Types* knowledge source contains a list of all the object types and features that have been incorporated in the domain model. For each object type, its name and properties are stored in this knowledge source. The properties of objects are: kernel, optional, variant, aggregate, agh_root, and gsh_root. The CLIPS implementation of this knowledge source is essentially a list of facts — one fact for each object type and its properties and one fact for each feature.

The various relationships and dependencies among features and between features and object types are captured in the *Inter-Feature & Feature-Object Dependencies* knowledge

source. The prerequisite relationship between two features is captured in a CLIPS fact with the key word "requires". For each feature, the object types required to support that feature are expressed as CLIPS facts using the key word "supported-by". These dependencies are enforced during feature selection or deletion by the *Dependency Checker* knowledge source.

The *Multiple Views* knowledge source contains the different views created using the EDLC methodology, in particular, the aggregation hierarchy and the generalization/ specialization hierarchies. These hierarchies are accessed and utilized by the *Target System Generator* knowledge source when the target system is being assembled. The parent-child relationship between objects in the aggregation hierarchy is expressed as CLIPS facts using the key word "is-part-of". The supertype-subtype relation between objects in the generalization/specialization hierarchy is expressed as CLIPS facts with the "is-a" key word.

GENERATION OF TARGET SYSTEM SPECIFICATION

To generate the target system specification, KBRET enters into a dialog with the target system engineer and elicits the requirements for the target system. A sample dialog, in which a domain model for the Payload Operations Control Center (POCC) application domain is used to generate a target system specification, is given in the Appendix.

At the start of the dialog, KBRET prints the system banner and asks the target system engineer whether he/she wishes to browse the domain model or would like to specify the requirements for the target system, as shown in the sample dialog in the Appendix. If the response is to browse, the browsing phase is initiated. The target system engineer can explore the domain model and get explanations for the different features incorporated in the domain model. Once sufficient familiarity with the domain model has been gained, the target system requirements specification phase may be initiated.

The target system engineer is presented with the various features captured in the domain model in the form of a menu, as shown in the Appendix, and the features desired in the target system can be selected from this menu. Whenever a feature is selected for the target system, the dependency checking phase is initiated, and the inter-feature and feature/object dependencies are checked and enforced. If a particular feature, say "F", requires the presence of other features, and they are not selected for the target system, the target system engineer is informed of that fact, and those features are automatically included in the target system in order to support feature "F".

An example of this feature dependency checking is shown in the sample dialog in the Appendix. When the target system engineer tries to select the Verifying Real Time Commands (feature 7), KBRET responds with a message saying that Verifying Real Time Commands requires Sending Real Time Commands feature and it will be automatically included in the target system, and requests the target system engineer's confirmation. When the target system engineer types "y" to confirm the selection, KBRET includes both the Sending Real Time Commands feature and the Verifying Real Time Commands feature in the target system and displays a message to that effect, as shown in the Appendix.

When a feature is selected for the target system, the object types that are required to support that feature are also selected in accordance with the feature/object dependencies and the CLIPS fact-base is updated to reflect that fact. The target system engineer, thus, can specify the requirements for the target system, and the *Feature & Object Selection/Deletion* knowledge source asserts new facts into the fact-base to record those selections. Of course, the *Dependency Checker* would ensure that the inter-feature and feature/object dependencies have not been violated.

The target system engineer can also delete features that have been selected for the target system. If a feature, say "F", is to be deleted, the *Dependency Checker* will check the fact-base to see if any of the features selected for the target system require that feature "F". If so, the deletion of feature "F" is disabled. An example of this deletion dependency checking is shown in the sample dialog. When the target system engineer tries to delete the Sending Real

Time Commands (feature 6) from the target system, KBRET responds with a message saying that the Sending Real Time Commands feature is required by the Verifying Real Time Commands feature and since Verifying Real Time Commands feature is currently selected for the target system, the Sending Real Time Commands feature cannot be deleted, and the dialog continues. When a feature "F" is deleted, it may cause the deletion of some other features if those features were included in the target system solely because of the selection of feature "F" and if they are not required by any other feature selected for the target system. The deletion of a feature also triggers the deletion of object types that were included to support that feature.

If the target system engineer would like to specify a feature that has not been captured in the domain model, the requirements elicitation phase is suspended and the domain analyst is called upon to model that requirement and enhance the domain model. Then, the target system specification and generation may be resumed.

Once the requirements for the target system have been completely specified, the target system generation phase is invoked. KBRET prompts for a name for the target system that is being generated so that it may be stored in the object repository for reuse. The fact-base is examined and the features and the object types selected for the target system are gathered. KBRET then presents the list of features that have been selected for the target system. The kernel object types are included in the target system because they must be part of every member of the family of systems. The selected variant and optional object types are examined to see if variant integration is required. If variant integration is not required, then the target system specification is generated and presented to the target system engineer.

In presenting the target system specification, KBRET provides two options. The target system engineer may view only the leaf-level object types or he/she can view both the aggregate and leaf-level object types. If the target system engineer chooses the second option, KBRET provides the aggregation hierarchy for the target system, as shown in the Appendix. This is accomplished by pruning the domain aggregation hierarchy, i.e., deleting from the domain aggregation hierarchy the object types that have not been included in the target system. KBRET presents the target system aggregation hierarchy in an indented form, as shown in the Appendix, to reflect the various levels of the aggregation hierarchy.

KBRET also outputs two files containing the target system information. These files are used to tailor the domain model graphical views and generate a set of graphical views for the target system. The target system views differ from those of the domain model in two ways. First, the optional objects that are not selected for the target system are removed. Secondly, in the case where one or more variants of a domain object type are selected, the object type is replaced by its variant(s).

If variant integration is required, the domain analyst is called upon to perform variant integration. When the integration process is completed, the target system generation phase is resumed and the target system specification is generated and presented to the target system engineer.

SUMMARY

This paper has discussed the domain modeling approach to software reuse and presented a prototype environment that supports domain modeling and generation of target system specifications. The architecture of the knowledge-based tool, KBRET, along with its domain-dependent and domain-independent knowledge sources was described. Finally, the paper has discussed KBRET's approach to generating target system specifications from the domain model, by eliciting the requirements for the target system and tailoring the domain model. A sample dialog with KBRET was also presented.

Future work includes extending KBRET to provide the capability to generate target system specifications from existing related target system specifications in conjunction with the domain model, and also to improve KBRET's user interface.

ACKNOWLEDGEMENTS

We gratefully acknowledge the assistance of S. Bailin, R. Dutilly, J. M. Moore, and W. Truszkowski in providing us with information on the POCC. We gratefully acknowledge the major contributions of Liz O'Hara-Schettino in developing the domain model of the POCC, and C. Bosch and I. Tavakoli for their major contributions to the prototype software engineering environment. This work was sponsored primarily by NASA Goddard Space Flight Center Automated Technology Branch Code 522.3 under the Code R research program, with support from the Virginia Center of Innovative Technology. The Software Through Pictures CASE tool was donated to GMU by Interactive Development Environments (IDE).

REFERENCES

Batory, D. (1989). The Genesis Database System Compiler: A Result of Domain Modeling. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Biggerstaff, T., Richter, C. (1987). Reusability Framework, Assessment, and Directions. *IEEE Software*, March 1987.

Giarratano, J.C. (1991). Clips User's Guide, Version 5.0, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX.

Gomaa, H. (1990). A Domain Analysis and Specification Method for Software Reuse. *Proc. Third Annual Workshop on Methods and Tools for Reuse*, Syracuse, NY.

Gomaa, H. (1992). An Object-Oriented Domain Analysis and Modeling Method for Software Reuse. *Proc. Hawaii International Conference on System Sciences*, Hawaii.

Gomaa, H., & Kerschberg, L. (1991a). An Evolutionary Domain Life Cycle Model for Domain Modeling and Target System Generation. *Proc. Workshop on Domain Modeling for Software Engineering, Int. Conf. on Software Engineering*, Austin, TX.

Gomaa, H., Fairley, R., and Kerschberg, L. (1989). Towards an Evolutionary Domain Life Cycle Model. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA*, New Orleans, LA.

Gomaa, H., Kerschberg, L., Bosch, C., Sugumaran, V., and Tavakoli, I. (1991b). A Prototype Software Engineering Environment for Domain Modeling and Reuse. *Proc. Fourth Annual Workshop on Methods and Tools for Reuse*. Herndon, VA.

Kang, K. C. et. al. (1990). Feature-Oriented Domain Analysis. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute.

Lubars, M. D. (1989). Domain Analysis for Multiple Target Systems. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Parnas, D. (1979). Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Eng.*, Vol. 5 No. 2, pp. 128-137.

Prieto-Diaz, R. (1987). Domain Analysis for Reusability. *Proc. of COMPSAC'87.*

Pyster, A. (1990). The Synthesis Process for Software Development. In R. Thayer and M. Dorfman (Eds.). *System and Software Requirements Engineering*, New York:IEEE Computer Society Press.

Sugumaran, V., Gomaa, H., and Kerschberg, L. (1991). Generating Target System Specifications from a Domain Model Using CLIPS. *Proc. of Second Annual Clips Conference*, Houston, TX.

Appendix. Sample Dialog with KBRET for the POCC domain.

```
************************************************************
*                                                          *
*  KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL           *
*                        (KBRET)                           *
*                                                          *
************************************************************
```

Requirements Elicitation for POCC domain
****

You may browse the features incorporated in the Domain Model, specify the requirements for the Target System or quit KBRET.

Choices	Perform
1	Browse the Domain Model
2	Specify requirements for Target System
3	Quit KBRET

Please type your choice and hit return: 1

Domain Model Browsing Phase
****

Please select one of the following choices to continue.

Choices	Perform
1	Explore the Features
2	Exit Browsing Phase
3	Quit KBRET

Please type your selection and hit return: 1

Feature Exploration
****

For the description of a feature, please type its number.

Choices	Feature to be described
1	Mission Type One
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	Exit Browsing Phase

Please type your selection and hit return: 5

Data Collection of Simulated Telemetry:
****

Simulated Telemetry Data can be collected and analyzed.

Choices	Feature to be described
1	Mission Type One
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands

```
        e       Exit Browsing Phase
Please type your selection and hit return: e

Exiting the Browsing Phase.....

You may browse the features incorporated in the Domain Model, or specify the requirements for the
Target System or quit KBRET.
    Choices         Perform
    *******         *******
        1       Browse the Domain Model
        2       Specify requirements for Target System
        3       Quit KBRET
Please type your choice and hit return: 2

        Target System Requirements Elicitation Phase
        ****************************************
Now, you will be presented with the features incorporated in the Domain Model. If a feature is desired in the
target system, please type its number and hit return. Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        1       Mission Type One
        2       Mission Type Two
        3       Experiment Type One
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 2
        The Mission Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        3       Experiment Type One
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 3
        The Experiment Type One Feature has been selected for the Target System.

Please select one of the following choices to continue.
    Choices         Feature to be selected
    *******         **********************
        4       Experiment Type Two
        5       Data Collection of Simulated Telemetry
        6       Sending Real Time Commands
        7       Verifying Real Time Commands
        e       End selecting features
Please type your selection and hit return: 4
        The Experiment Type Two Feature has been selected for the Target System.
```

Please select one of the following choices to continue.

Choices	Feature to be selected
5	Data Collection of Simulated Telemetry
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End selecting features

Please type your selection and hit return: 7

The Verifying Real Time Commands Feature requires Sending Real Time Commands Feature.
The Sending Real Time Commands Feature will be automatically included if the Verifying Real Time Commands Feature is desired.

Please type 'y' to include or 'n' to not include the Sending Real Time Commands Feature. (y/n): y

The Sending Real Time Commands Feature has been selected for the Target System.
The Verifying Real Time Commands Feature has been selected for the Target System.

Please select one of the following choices to continue.

Choices	Feature to be selected
5	Data Collection of Simulated Telemetry
e	End selecting features

Please type your selection and hit return: e

Target System feature selection has been exited...

The following features are currently selected:

Number	Feature Name
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands

Please select one of the following choices to continue.

Choices	Perform
1	Select more features for Target System
2	Delete a feature from Target System
3	Specify features not in the Domain Model
4	Initiate Target System Generation Phase
5	Quit KBRET

Please type your selection and hit return: 2

Please select one of the following choices to continue.

Choices	Feature to be deleted
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End deleting features

Please type your selection and hit return: 6

Since the Sending Real Time Commands Feature is required by the Verifying Real Time Commands Feature and since the Verifying Real Time Commands Feature is currently desired in the Target System, the Sending Real Time Commands Feature may not be deleted now.

Please type (c) and hit return to continue: c

Please select one of the following choices to continue.

Choices	Feature to be deleted
2	Mission Type Two
3	Experiment Type One
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End deleting features

Please type your selection and hit return: 3

Since Experiment Type One Feature is not required by any other target system feature, it will be deleted from the Target System Features.

Please type 'y' to delete or 'n' to abort the deletion of Experiment Type One Feature (y/n) : y

The Experiment Type One Feature has been deleted from the Target System.

Please select one of the following choices to continue.

Choices	Feature to be deleted
2	Mission Type Two
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands
e	End deleting features

Please type your selection and hit return: e

Target System feature deletion has been exited...

The following features are currently selected:

Number	Feature Name
2	Mission Type Two
4	Experiment Type Two
6	Sending Real Time Commands
7	Verifying Real Time Commands

Please select one of the following choices to continue.

Choices	Perform
1	Select more features for Target System
2	Delete a feature from Target System
3	Specify features not in the Domain Model
4	Initiate Target System Generation Phase
5	Quit KBRET

Please type your selection and hit return: 4

Invoking the Target System Generation Phase.....

Target System Generation Phase:

Please input a name for the Target System: EXAMPLE

EXAMPLE Target System Components

The following features have been selected for the EXAMPLE Target System

Mission Type Two Feature

Experiment Type Two Feature
Sending Real Time Commands Feature
Verifying Real Time Commands Feature

Assembling the EXAMPLE Target System. Please Wait.....

The Target System Object Types have been assembled. To view those object types included in the Target System, Please select one of the following choices:

Choices	Perform
*******	*******
1	View Leaf Level Object Types
2	View Aggregate and Leaf Level Object Types

Please type your selection and hit return: 2

The Aggregate and Leaf Level Objects of the Target System:
****

- Payload Operations Control Center Domain (kernel aggregate)
  - Telemetry (kernel aggregate)
    - Telemetry Pre-Processor (kernel)
    - Spacecraft Telemetry Processor (kernel aggregate)
      - Mission Two SC Eng. Telemetry Analog Limits Checker With Eqn. Processing (variant)
      - Mission Two SC Engineering Telemetry Trend Analyzer (variant)
      - Mission Two SC Engineering Telemetry Equation Processor (variant)
      - Mission Two Discrete SC Engineering Telemetry Analyzer (variant)
      - Mission Two FDF Interface (variant)
    - Observatory Telemetry Processor (kernel aggregate)
      - Experiment Two Instrument Telemetry Analog Limits Checker (variant)
      - Experiment Two Instrument Telemetry Trend Analyzer (variant)
      - Experiment Two Discrete Instrument Telemetry Analyzer (variant)
      - Experiment Two Scientific Telemetry Analyzer (variant)
    - TAC Interface (kernel)
    - RUPS Interface (kernel)
  - Command (kernel aggregate)
    - Command Load Processor (kernel aggregate)
      - Satellite Bound Command Load Processor (kernel)
      - Earth Bound Command Load Verifier (kernel)
      - Command Load Data Store (kernel)
      - OBC Image Verifier (kernel)
      - CMS Interface (kernel)
    - Real Time Command Processor (optional aggregate)
      - Satellite Bound Real-Time Command Processor (optional)
      - Earth Bound Real-Time Command Verifier (optional)
      - Real-Time Command Data Store (optional)
    - Satellite Bound Command Problem Resolver (optional)
  - Flight Operations Analyst (kernel aggregate)
    - FOA STOL Interface (kernel)
    - POCC Mode Selector (kernel)
    - FOA Command Processor (kernel)
    - FOA NCC Processor (kernel)
    - FOA Telemetry Processor (kernel)
    - NCC Interface (kernel)
  - History (kernel aggregate)
    - Telemetry History (kernel)
    - Command History (kernel)
    - Flight Operations Analyst History (kernel)
    - Telemetry Block History (kernel)

The EXAMPLE Target System Generation is complete. The object types shown above have been included in it and no variant integration is required.